Chapter 3

Readout Controller Library

3.1 Overview

The Readout Controller (formally known as the MVME6100) is a VMEbus single-board computer (SBC) with a powerful MPC7457 PowerPC processor running at 1.267 GHz. Combined with being the first series of SBCs to run 2eSST (two edge source synchronous transfers), the MVME6100 series delivers unprecedented data acquisition power for embedded system applications.

For our particular application, we have implemented VxWorks by Wind River as the operating system for the ROC (Readout Controller). The VxWorks OS can be found in numerous applications including Linksys WRT54G routers (version 5 and later), Boeing 787 and 747-8 aircraft, as well as many spacecraft including Spirit and Opportunity, the Mars Exploration Rovers.

Unlike other self-contained operating systems, VxWorks must have its programs cross-compiled on a separate operating system such as Linux. A description of the algorithms used for the Region 1 detector as well as how to compile these programs are contained within this chapter.

3.2 V1495 User Programs For The ROC

3.2.1 v1495Init

This module establishes a pointer value for the V1495 VME card within the MVME6100 memory map; the VxWorks documentation refers to this as a "memory anchor." Without this, all other v1495 programs on the ROC will not know how to function properly since the location of the V1495 card will not be known to the ROC. To establish a connect between the ROC and the V1495, one needs to provide the 32-bit VME address of the V1495 as an argument to the function. The first 16 bits of the 32-bit VME address are physically set on the V1495 card using the hex switches as shown in Figure 3-1 below. Here we have chosen the VME address as 0x08110000.

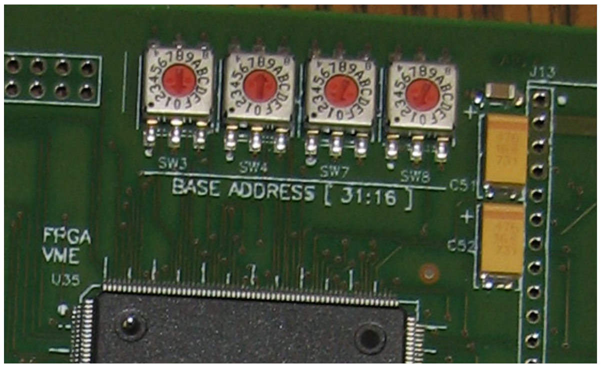

Figure 3-1: Hexidecimal Address Switches on V1495

Before the memory map address on the ROC is actually determined, however, this function checks to make sure that 32-bit addressing is supported on the ROC by making sure that the VxWorks version does not correspond to VXWORKS68K51: otherwise an error is thrown and a link will not be established.

Then, in order to actually establish a memory location in the ROC, a call to the VxWorks function "sysBustoLocalAddr" is made.

For all v1495 functions, with the exception of v1495Init, verification is made to make sure that a link has been established between the ROC and the V1495 card before performing any other functionality. This check will not be mentioned in any of the other functions' respective sections but the reader should be aware of the presence of this check.

3.2.2 v1495ReadEvent

The v1495ReadEvent function queries the VFATs and then stores their data into the "V1495ReadoutStatus" struct before getting parsed up and sent to the DAQ.

The following values are stored into the "V1495Readout Status struct":

- TotalBytes (Contains the total number of bytes recorded from all of the VFATs)
- TotalFrames (Contains the number of events/frames contained in all of the VFATs)
- DeltaBytes (In its current form, this performs exactly as "TotalBytes")
- DeltaFrames (In its current form, this performs exactly the same as TotalFrames)
- LastSize (This records the value for the most recent number of words in the DataFIFO for the last event.)
- LengthDataFIFO (Contains the most recent number of words in the DataFIFO)
- LengthSizeFIFO (Contains the most recent number of words in the SizeFIFO)
- Sent (Contains the total number of events sent per VFAT)
- LastCapture[12] (Contains the actual DataOut data from the VFATs)

3.2.3 v1495FillData

This function simply takes a pointer to a 32-bit integer and fills it with the data from the structs of type "V1495ReadoutStatus" with 32-bit "chunks." Here "chunks" refers to the fact that the data elements in the "V1495ReadoutStatus" structs are themselves not 32-bit elements; thus the sizes of the data elements are lost in this transition. The position and size of the data elements are just simply known on the receiving end of the data in the CODA data acquisition system.

This array of 32-bit pointers is filled out by first calling "v1495ReadEvent" to get the number of words in the event data if there exists at least one event recorded on the V1495 and then subsequently filling out the "V1495ReadoutStatus" structs.

A more detailed description of the v1495ReadEvent function can be found in that respective section.

3.2.4 v1495Sprint

This function is a watered-down printout of some of the most useful registers recorded by the V1495 after a query of the VFAT data.

While not normally used in regular operation, this function can provide extremely useful information regarding VFAT operation; especially during debugging.

The registers printed off by this function include one of the following for each VFAT:

- Number of words in dataFIFO (dataFIFO is the FIFO containing the actual data from the VFAT)
- Number of words in sizeFIFO (sizeFIFO is the FIFO containing the number of words contained in the dataFIFO)
- EVENTSSENTH (upper 16 bits from the number of events currently recorded in the V1495 with an “event” being the number of times the DataValid line on the VFAT goes from high to low)
- EVENTSSENTL (lower 16 bits from the number of events currently recorded in the V1495)
- HARD_TRIGGER_WORD (the 4-bit hexadecimal value sent out on the T1 line whenever a HARD TRIGGER pulse is sent to the G1 port on the V1495; by default this is 0x4)

3.2.5 v1495StatusPrint

As with v1495Sprint, this function is not used during regular operation, but its output includes many useful values for debugging among which are the following for each VFAT:

- Number of TotalBytes
- Number of TotalFrames
- Number of Sent Bytes
- VFAT serial data in hexadecimal, 32-bit segments

If there is serial data present in the V1495 then the EventID as well as the ChipID for each VFAT will also be printed.

3.2.6 v1495HextoBin

As the name implies, this function simply changes the hexadecimal values reported in the v1495StatusPrint function to their binary equivalent. This can make it easier to determine which lines on the VFAT recorded a hit.

It must be noted, however, that this is a computationally intensive function and it has been found to crash the ROC while the CODA system is running, so this basically reduces it functionality to debugging.

3.2.7 v1495test

This function prints off several key registers found in the V1495. Each of these is useful on a VME card-by-card basis and could quite possibly be used to identify a particular V1495 card if more than one is used. The registers that are printed out are the following:

- Control Register
- Firmware Revision
- SelfflashVME
- Flash VME
- Self Flash User
- flash USER
- Configuration ROM

3.2.8 v1495reload

The v1495reload function is called from within v1495firmware and simply reboots the User FPGA by writing a '1' to the USER FPGA Configuration Register. This is a crucial step as the USER FPGA will maintain its original configuration unless this is done.

3.2.9 v1495firmware

While the purpose of the V1495 User Firmware is discussed in the "V1495 Module by CAEN" chapter, there are still several ways to actually accomplish the downloading process for the firmware.

One can use the precompiled software for Windows from CAEN as described in the V1495 User Manual, or one can use the VME backplane along with the V1495 Bridge FPGA to flash the User FPGA.

Using the latter method, the “v1495firmware” function was actually developed by a third party, namely C. Tintori of CAEN, but its overarching functionality is simple enough and to not use it carefully can have adverse effects such as accidentally overwriting the FPGA VME firmware when it was intended to overwrite the FPGA User firmware. The UML diagram for the firmware downloading process is shown below in Figure ?? and is originally found on page 45 of the V1495 User Manual.

The first parameter passed to this function is the ROC memory address for the V1495, e.g. 0x80110000 for our particular V1495. The second is the actual firmware file   enclosed in quotes as well   which should have the extension “.rbf” is generated using Quartus II. The third parameter tells the function whether to write to the “standard” image or the “backup” image. To accomplish this a ‘0’ needs to be passed for the “standard” image and a ‘1’ for the “backup” image. It must be noted that one cannot write a “backup” image to the User FGPA. The fourth and last parameter to this function tells the function whether to download to the User FPGA or to the VME FPGA; either a ‘0’ or ‘1’ respectively.

3.2.10 v1495release

This function simply sets the v1495 pointer to a NULL value thereby releasing the reference of the “V1495ReadoutCtrlRegs“ struct to the ROC memory map.

3.2.11 v1495DataReady

This function uses the FIFOLength register to determine which VFAT has the greatest number of entries. The greatest number of entries is then returned as an argument by the function.

3.2.12 v1495CalPulse

This function was originally intended to take the parameter passed to it to instruct the V1495 to send that number of CalPulses to the VFATs.

In its current form, this function can perform one of two tasks: Sending a number other than

0xFFFF will instruct the V1495 to send a CalPulse on the next MCLK signal; if 0xFFFF is sent then every time an external Hard Trigger signal is send, rather than sending a LV1A signal the V1495 sends a CalPulse signal. To have the V1495 return to normal operation one only needs to send a number other than 0xFFFF with this function.

3.2.13 v1495Reset

This function simply resets the V1495. It accomplishes this by writing a ‘0’ to the reset register in the V1495. For more information on how this is accomplished see the chapter “V1495 Module by CAEN”.

3.2.14 Other VxWorks Debugging Programs

m <32-bit address>,<Number of Bytes>

The ‘m’ command which stands for “modify” is a particularly useful, native VxWorks command that helps one to write to a given register address. The syntax for this function is given above. After the function enter the beginning address to be written to followed by how many bytes of data you wish to change. Hit Enter if you don’t want to change a value. Enter a ‘.’ and hit enter to exit the function.

d <32-bit address>,<Number of Bytes>

The ‘d’ command which stands for “dump” displays the values for the requested number of bytes. Enter a ‘.’ to exit the function. Note that this function displays the value of the bytes on the screen relative to their hexadecimal address: therefore, addresses ending in 0x0 display to the far left while addresses ending in 0xF to the far right with all others displaying in their respective, relative position.

Also, I was not able to find in any literature I had found what the characters next to the function are that appear between the two asterisks. I suspect that it may have to do with error correction, but this is of course just a guess.

3.3 Compiling programs for usage on the ROC

As previously discussed, the ROC uses the VxWorks operating system which cannot compile its own programs: it is therefore necessary to cross-compile its programs using another operating system such as Linux. As with most real-world applications, the software packages necessary to cross-compile our programs come pre-made and we have no exception here. The “makefile” used to cross-compile our programs is included in the appendix.